Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

за 1 семестр

По дисциплине: «Языки программирования»

Тема: «Перезагрузка операций и принципов обработки исключений»

Вариант №7.

Выполнила:

Савицкая А.Ю., ПО-7

Проверил:

Бойко Д.О.

Брест, 2021

**Цель работы:** получить практические навыки создания иерархии классов и использования статических компонентов класса.

**Задание:** Написать программу, в которой описана иерархия классов: ошибка в программе («недостаточно привилегий», «ошибка преобразования», «невозможно преобразовать значение»). Описать класс для хранения коллекции ошибок (массива указателей на базовый класс), в котором перегрузить операцию «[ ]». Для базового класса и его потомков перегрузить операции «==», «!=», «=». Продемонстрировать работу операторов.

**Ход работы:**

**1) Иерархия классов:**



**2) Текст программы:**

**1. Software\_Errors.h**

#pragma once

#include <iostream>

using namespace std;

class Software\_Errors {

public:

virtual void Print() = 0;

virtual void Read() = 0;

};

**2. sSoftware\_Errors.h**

#pragma once

#include "Software\_Errors.h"

#include "error.h"

class sSoftware\_Errors {

private:

Software\_Errors\*\* arr;

int count;

public:

sSoftware\_Errors(int count);

~sSoftware\_Errors();

Software\_Errors\* operator[] (int n) const;

Software\_Errors\*& operator[] (int n);

int get\_count();

void addToTheEnd(Software\_Errors\* figure);

void add(int index, Software\_Errors\* figure);

void deleteFromTheEnd();

void del(int index);

};

**3. sSoftware\_Errors.cpp**

#include "sSoftware\_Errors.h"

sSoftware\_Errors::sSoftware\_Errors(int count) {

arr = new Software\_Errors \* [count];

for (int i = 0; i < count; i++) {

arr[i] = nullptr;

}

this->count = count;

}

sSoftware\_Errors::~sSoftware\_Errors() {

for (int i = 0; i < count; i++) {

if (arr[i] != nullptr) {

delete arr[i];

}

delete[]arr;

}

}

Software\_Errors\* sSoftware\_Errors::operator[] (int n) const {

if (n < 0 || n >= count) {

throw IndexError("Вы вышли за границы массива.");

}

return arr[n];

}

Software\_Errors\*& sSoftware\_Errors::operator[] (int n) {

if (n < 0 || n >= count) {

throw IndexError("Вы вышли за границы массива.");

}

return arr[n];

}

int sSoftware\_Errors::get\_count() {

return count;

}

void sSoftware\_Errors::addToTheEnd(Software\_Errors\* figure) {

Software\_Errors\*\* temp = new Software\_Errors \* [count + 1];

for (int i = 0; i < count; i++) {

temp[i] = arr[i];

}

arr = temp;

temp[count] = figure;

count++;

}

void sSoftware\_Errors::add(int index, Software\_Errors\* figure) {

if (index < 0 || index > count) {

addToTheEnd(figure);

throw IndexError("Вы вышли за границы массива. Ошибка будет добавлена в конец массива");

}

Software\_Errors\*\* temp = new Software\_Errors \* [count + 1];

for (int i = 0; i < index; i++) {

temp[i] = arr[i];

}

temp[index] = figure;

for (int i = index; i < count; i++) {

temp[i + 1] = arr[i];

}

arr = temp;

count++;

}

void sSoftware\_Errors::deleteFromTheEnd() {

Software\_Errors\*\* temp = new Software\_Errors \* [count - 1];

for (int i = 0; i < count - 1; i++) {

temp[i] = arr[i];

}

arr = temp;

count--;

}

void sSoftware\_Errors::del(int index) {

Software\_Errors\*\* temp = new Software\_Errors \* [count - 1];

for (int i = 0; i < index; i++) {

temp[i] = arr[i];

}

for (int i = index + 1; i < count; i++) {

temp[i - 1] = arr[i];

}

arr = temp;

count--;

}

**4. Insufficient\_Privileges.h**

#pragma once

#include "Software\_Errors.h"

class Insufficient\_Privileges :public Software\_Errors {

private:

int kol\_errors;

public:

Insufficient\_Privileges();

Insufficient\_Privileges(int kol\_errors);

Insufficient\_Privileges(const Insufficient\_Privileges& other);

~Insufficient\_Privileges();

bool operator == (const Insufficient\_Privileges& right);

bool operator != (const Insufficient\_Privileges& right);

Insufficient\_Privileges& operator = (const Insufficient\_Privileges& right);

void Print();

void Read();

};

**5. Insufficient\_Privileges.cpp**

#include "Insufficient\_Privileges.h"

Insufficient\_Privileges::Insufficient\_Privileges() {}

Insufficient\_Privileges::Insufficient\_Privileges(int kol\_errors) {

this->kol\_errors = kol\_errors;

}

Insufficient\_Privileges::Insufficient\_Privileges(const Insufficient\_Privileges& other) {

kol\_errors = other.kol\_errors;

}

Insufficient\_Privileges::~Insufficient\_Privileges() {}

bool Insufficient\_Privileges::operator == (const Insufficient\_Privileges& right) {

return (kol\_errors == right.kol\_errors);

}

bool Insufficient\_Privileges::operator != (const Insufficient\_Privileges & right) {

return !(\*this == right);

}

Insufficient\_Privileges& Insufficient\_Privileges::operator = (const Insufficient\_Privileges & right) {

kol\_errors = right.kol\_errors;

return \*this;

}

void Insufficient\_Privileges::Print() {

cout << "Ошибка - ошибка недостатка привелегий" << endl << "Количество ошибок:" << kol\_errors << endl;

}

void Insufficient\_Privileges::Read() {

cout << "Введите количество ошибок: ";

cin >> kol\_errors;

}

**6. Conversion\_Error.h**

#pragma once

#include "Software\_Errors.h"

class Conversion\_Error :public Software\_Errors {

private:

int pop;

public:

Conversion\_Error();

Conversion\_Error(int pop);

Conversion\_Error(const Conversion\_Error& other);

~Conversion\_Error();

bool operator == (const Conversion\_Error& right);

bool operator != (const Conversion\_Error& right);

Conversion\_Error& operator = (const Conversion\_Error& right);

void Print();

void Read();

};

**7. Conversion\_Error.cpp**

#include "Conversion\_Error.h"

Conversion\_Error::Conversion\_Error() {}

Conversion\_Error::Conversion\_Error(int pop) {

this->pop = pop;

}

Conversion\_Error::Conversion\_Error(const Conversion\_Error& other) {

pop = other.pop;

}

Conversion\_Error::~Conversion\_Error() {}

bool Conversion\_Error::operator == (const Conversion\_Error& right) {

return (pop == right.pop);

}

bool Conversion\_Error::operator != (const Conversion\_Error & right) {

return !(\*this == right);

}

Conversion\_Error& Conversion\_Error::operator = (const Conversion\_Error & right) {

pop = right.pop;

return \*this;

}

void Conversion\_Error::Print() {

cout << "Ошибка - ошибка преобразования" << endl << "Количестов попыток устранения ошибки:" << pop << endl;

}

void Conversion\_Error::Read() {

cout << "Введите количестов попыток устранения ошибки: ";

cin >> pop;

}

**8. Cannot\_Convert\_Value.h**

#pragma once

#include "Software\_Errors.h"

class Cannot\_Convert\_Value :public Software\_Errors {

private:

int dif;

public:

Cannot\_Convert\_Value();

Cannot\_Convert\_Value(int dif);

Cannot\_Convert\_Value(const Cannot\_Convert\_Value& other);

~Cannot\_Convert\_Value();

bool operator == (const Cannot\_Convert\_Value& right);

bool operator != (const Cannot\_Convert\_Value& right);

Cannot\_Convert\_Value& operator = (const Cannot\_Convert\_Value& right);

void Print();

void Read();

};

**9. Cannot\_Convert\_Value.cpp**

#include "Cannot\_Convert\_Value.h"

Cannot\_Convert\_Value::Cannot\_Convert\_Value() {}

Cannot\_Convert\_Value::Cannot\_Convert\_Value(int dif) {

this->dif = dif;

}

Cannot\_Convert\_Value::Cannot\_Convert\_Value(const Cannot\_Convert\_Value& other) {

dif = other.dif;

}

Cannot\_Convert\_Value::~Cannot\_Convert\_Value() {}

bool Cannot\_Convert\_Value::operator == (const Cannot\_Convert\_Value& right) {

return (dif == right.dif);

}

bool Cannot\_Convert\_Value::operator != (const Cannot\_Convert\_Value & right) {

return !(\*this == right);

}

Cannot\_Convert\_Value& Cannot\_Convert\_Value::operator = (const Cannot\_Convert\_Value & right) {

dif = right.dif;

return \*this;

}

void Cannot\_Convert\_Value::Print() {

cout << "Ошибка - невозможно преобразовать значение" << endl << "Сложность ошибки (от 0 до 10):" << dif << endl;

}

void Cannot\_Convert\_Value::Read() {

cout << "Оцените сложность ошибки (от 0 до 10 ): ";

cin >> dif;

}

**10. error.h**

#pragma once

#include <iostream>

class IndexError {

protected:

char\* message;

public:

IndexError(const char\* message);

~IndexError();

char\* get\_message();

};

**11. error.cpp**

#include "error.h"

IndexError::IndexError(const char\* message) {

this->message = \_strdup(message);

}

IndexError::~IndexError() {

delete this->message;

}

char\* IndexError::get\_message() {

return message;

}

**12. main.cpp**

#include "Cannot\_Convert\_Value.h"

#include "Conversion\_Error.h"

#include "Insufficient\_Privileges.h"

#include "sSoftware\_Errors.h"

#include "error.h"

void menu(sSoftware\_Errors arr);

int main() {

setlocale(LC\_ALL, "ru");

system("color f0");

cout << "Задание. Вариант 7." << endl

<< "Написать программу, в которой описана иерархия классов: ошибка в программе(«недостаточно привилегий», «ошибка преобразования», «невозможно преобразовать значение», «невозможно привести к интерфейсу»)."

<< endl << "Описать класс для хранения коллекции ошибок(массива указателей на базовый класс), в котором перегрузить операцию «[]»."

<< endl << "Для базового класса и его потомков перегрузить операции « == », « != », « = ».Продемонстрировать работу операторов." << endl;

sSoftware\_Errors arr(0);

menu(arr);

return 0;

}

void menu(sSoftware\_Errors arr) {

bool exit = false;

while (!exit) {

cout << "1 - добавить ошибку недостатка привилегий" << endl

<< "2 - добавить ошибку преобразования" << endl

<< "3 - добавить ошибку невозможности преобразовать значение" << endl

<< "4 - сравнить ошибки недостатка привелегий" << endl

<< "5 - показать ошибку по индексу" << endl

<< "6 - показать все ошибки" << endl

<< "7 - удалить с конца" << endl

<< "8 - удалить по индексу" << endl

<< "0 - Exit" << endl;

try

{

int num;

cin >> num;

cout << endl;

switch (num)

{

case 1:

{

Insufficient\_Privileges\* obj1 = new Insufficient\_Privileges(0);

obj1->Read();

Insufficient\_Privileges\* obj2 = new Insufficient\_Privileges(0);

obj2 = obj1;

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj2);

}

else {

arr.addToTheEnd(obj2);

}

break;

}

case 2:

{

Conversion\_Error\* obj = new Conversion\_Error(0);

obj->Read();

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj);

}

else {

arr.addToTheEnd(obj);

}

break;

}

case 3:

{

Cannot\_Convert\_Value\* obj = new Cannot\_Convert\_Value(0);

obj->Read();

bool temp = false;

cout << "Добавить в конец(0) или по индексу(1)?";

cin >> temp;

if (temp) {

int index;

cout << "Введите индекс: ";

cin >> index;

arr.add(index, obj);

}

else {

arr.addToTheEnd(obj);

}

break;

}

case 4:

{

Insufficient\_Privileges obj1(0);

obj1.Read();

Insufficient\_Privileges obj2(0);

obj2.Read();

if (obj1 == obj2) {

cout << "Ошибки равны" << endl;

}

else if (obj1 != obj2) {

cout << "Ошибки не равны" << endl;

}

break;

}

case 5:

{

int index;

cout << "Введите индекс: ";

cin >> index;

if (arr[index] != 0) {

arr[index]->Print();

}

break;

}

case 6:

{

cout << "Количество всех ошибок = " << arr.get\_count() << endl;

for (int i = 0; i < arr.get\_count(); i++) {

arr[i]->Print();

}

break;

}

case 7:

{

arr.deleteFromTheEnd();

break;

}

case 8:

{

int index;

cout << "Введите индекс: ";

cin >> index;

arr.del(index);

break;

}

case 0:

exit = true;

}

cout << endl;

}

catch (IndexError & e)

{

cout << "Ошибка индекса: " << e.get\_message() << endl;

}

catch (...)

{

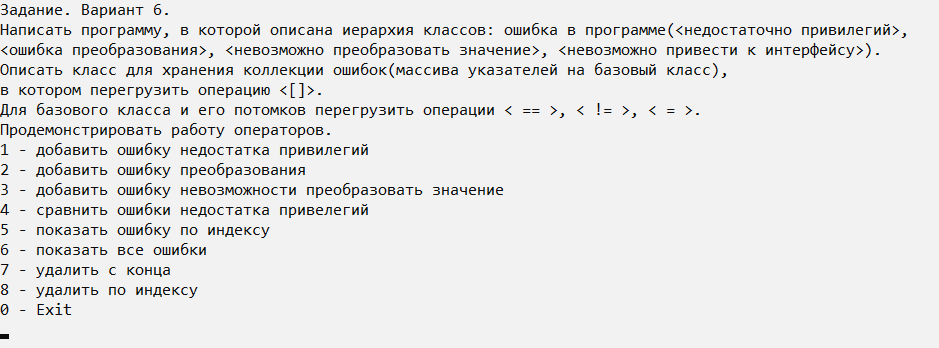
cout << "Неизвестная ошибка." << endl;

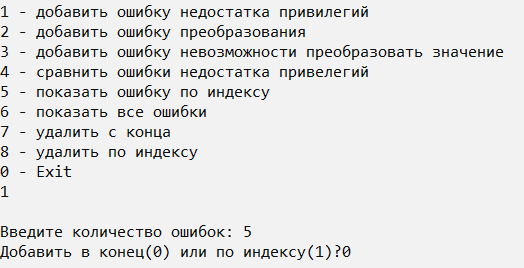
}

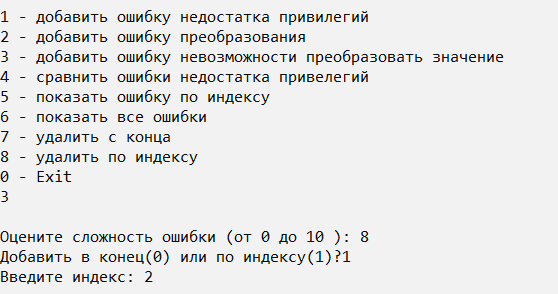
}

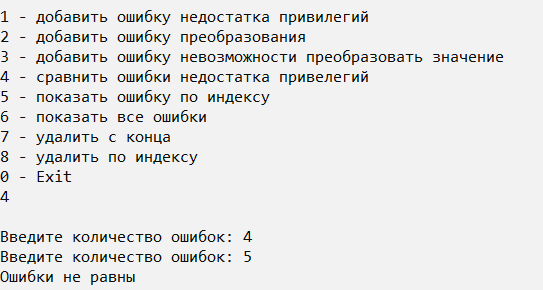
}

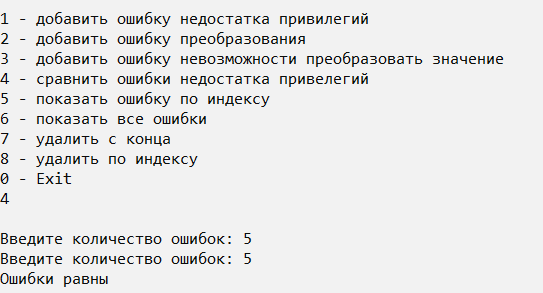
**3) Результаты тестирования программы:**

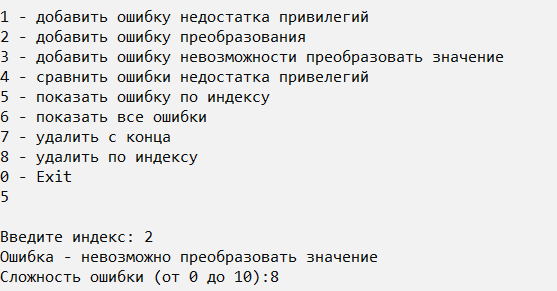
****

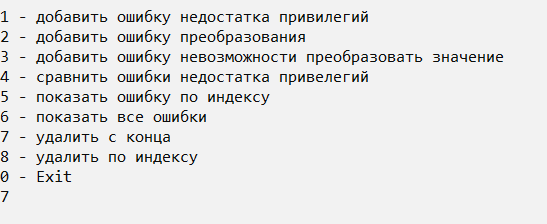


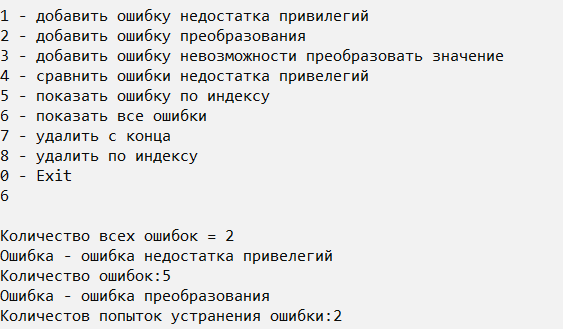


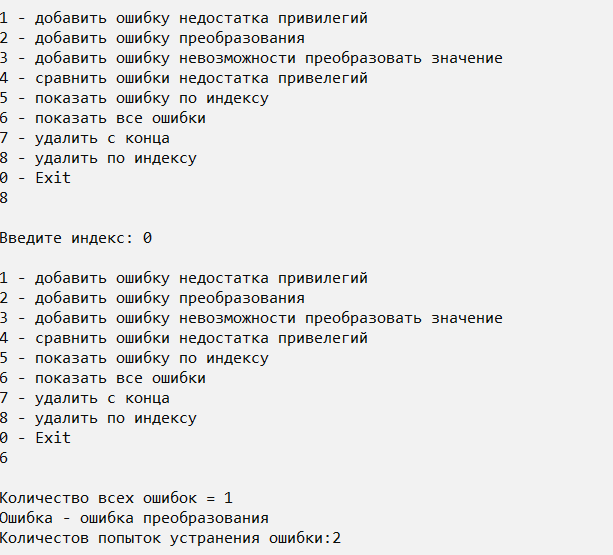












**Вывод:** получила практические навыки создания иерархии классов и использования статических компонентов класса.